End-to-end quality of service for large distributed storage

Scott A. Brandt
Professor
University of California, Santa Cruz

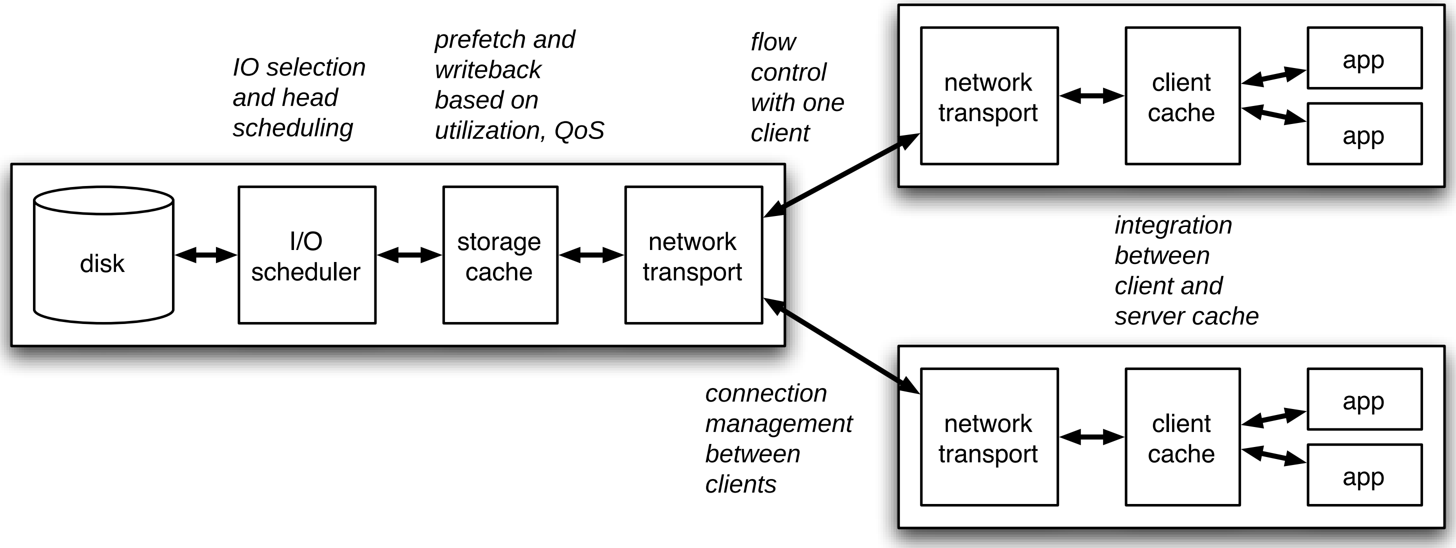
and Carlos Maltzahn, Richard Golding, Theodore Wong
and Tim Kaldewey, Roberto Pineiro, Anna Povzner

6 August 2007

Project overview

- Collaboration between UCSC / IBM Almaden
  - UCSC: Scott Brandt, Carlos Maltzahn
  - IBM: Richard Golding, Theodore Wong
  - 3 years / $1,000,000
- Goal: Improve end-to-end performance management in large clustered storage
  - From client, through server, to disk
  - Manage performance
  - Isolate traffic
  - Provide high performance

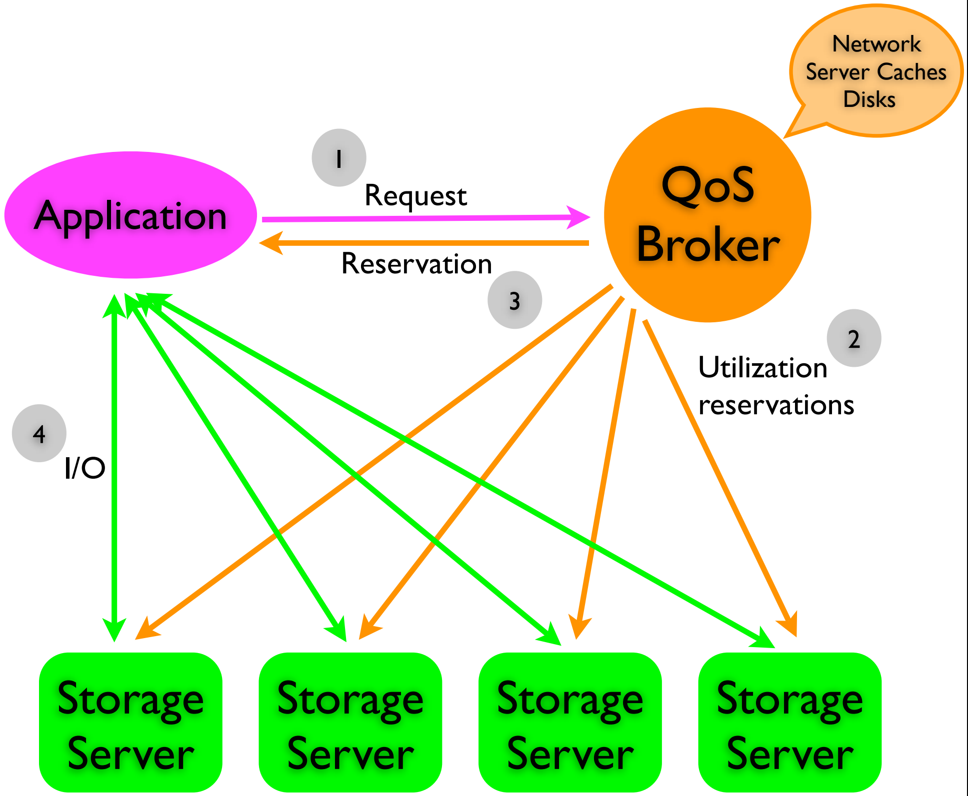
Stages in the I/O path



1. Disk traffic
2. Management of server cache
3. Flow control across network
    - within one client's session; between clients
4. Management of client cache

System architecture

- Applications request reservation from broker
    - Specify workload: throughput, read/write ratio, burstiness, etc.
- Broker does admission control
    - Requirements are translated to utilization
    - Utilizations are summed to see if they are feasible
    - Once admitted, I/O streams are guaranteed (subject to workload adherence)
- Disk, cache, network controllers maintain guarantees



Fahrrad: Efficient QoS-aware Disk Scheduling

- Control of application resource reservation and usage *at the disk level*

- Goals:
  - Mixed hard, soft, and non-real-time workloads
  - Arbitrary granularity of reservations
  - Complete isolation of workloads
  - Excellent I/O performance

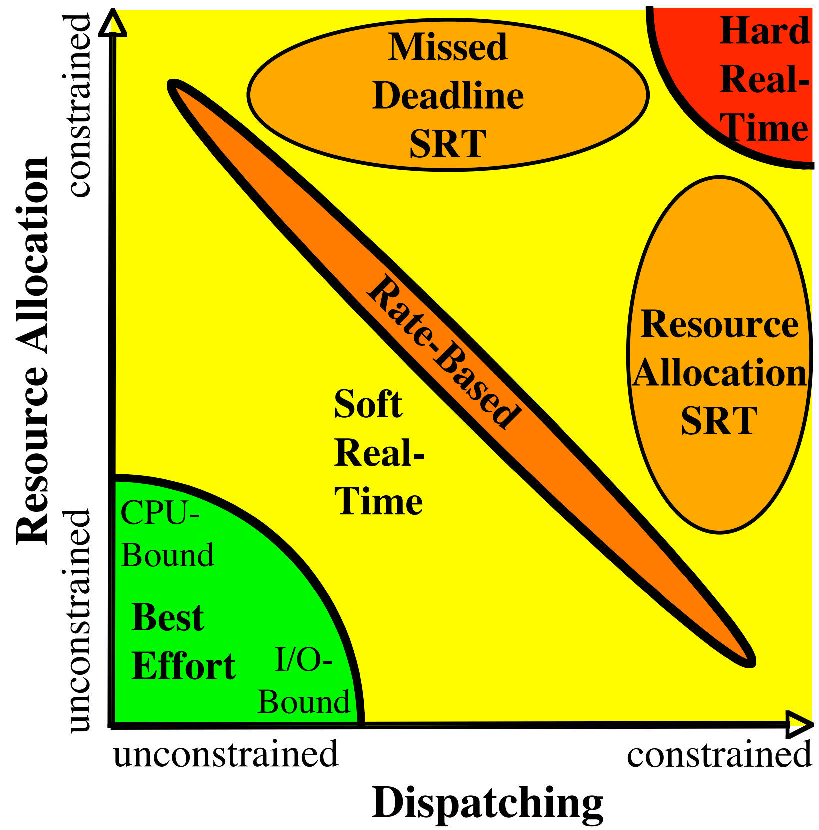
Key observation

- Scheduling consists of two distinct questions

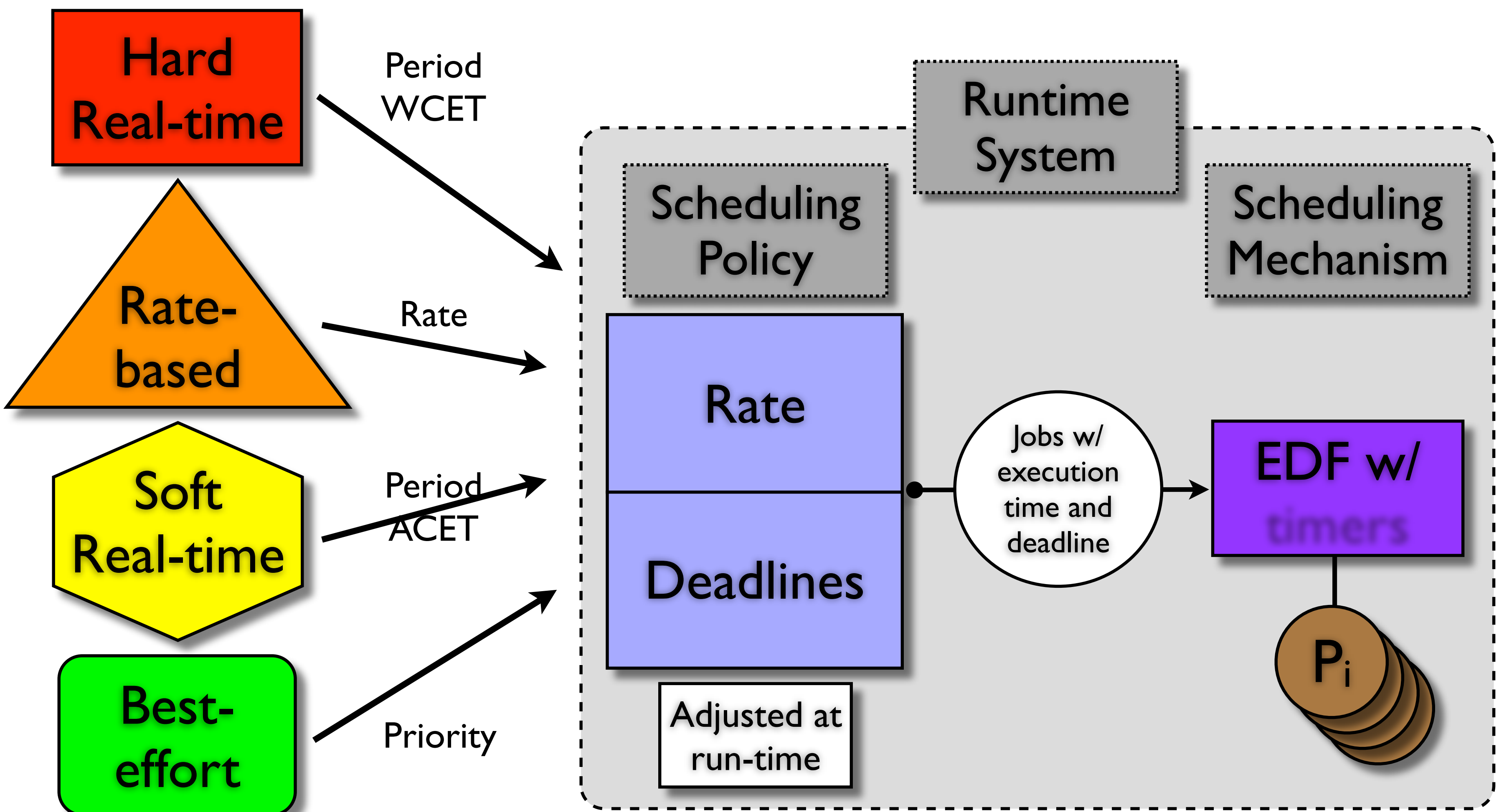
  Resource allocation: *How much* resources to allocate to each process

  Dispatching: *When* to give each process the resources it has been allocated

- Most schedulers integrate their management
- Separating them is powerful!



RBED RAD-based CPU scheduler



Utilization-based disk reservations

- Throughput reservations
  - Assume worst-case behavior
  - Allows reservation of a tiny fraction of actual throughput
- Utilization reservations
  - Easy to make, account for, and guarantee
  - Embed application workload information
  - Avoid the need for worst-case assumptions
- Workload knowledge + utilization reservation + isolation = throughput guarantee

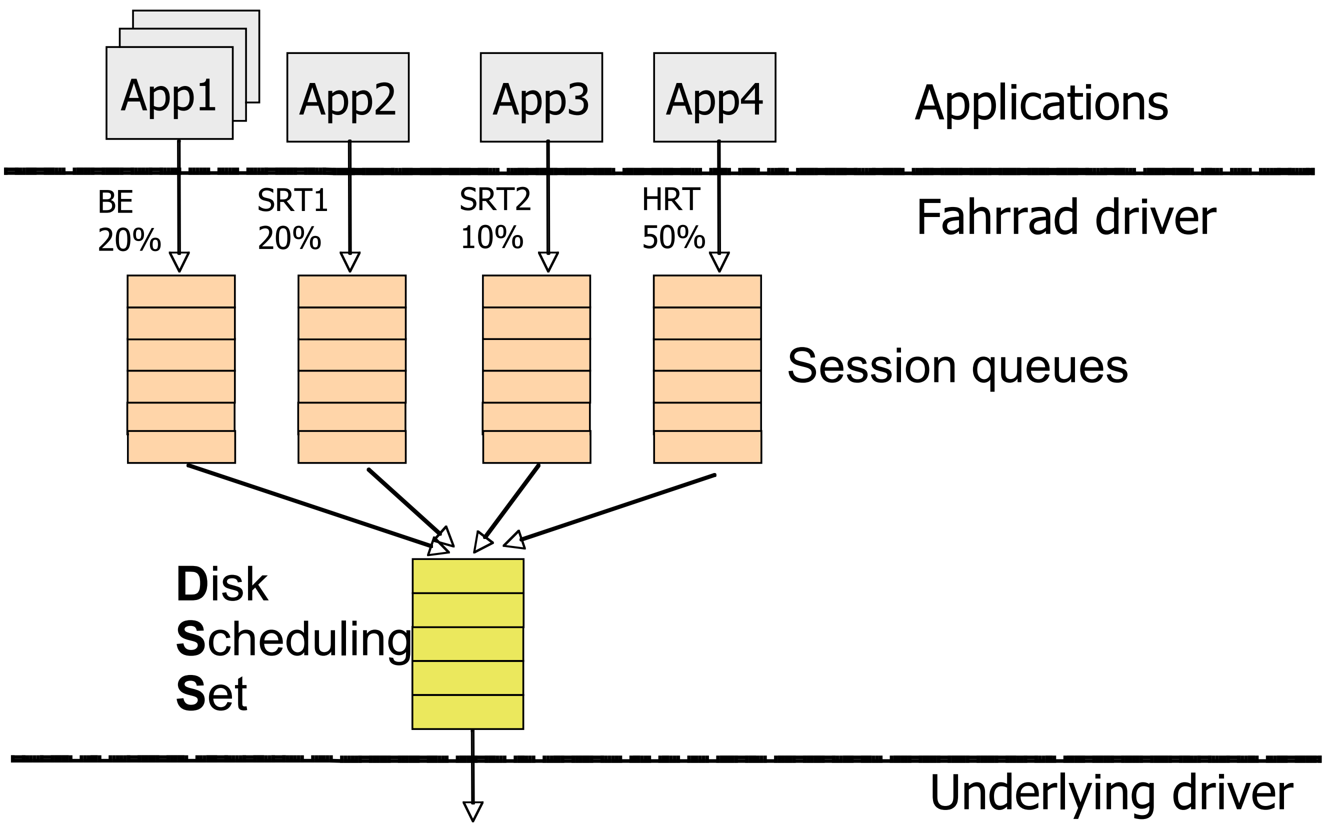
Applying RAD to disk I/O

- Reservations based on disk time utilization
  - Rate = utilization
  - Deadlines = times at which actual utilization must equal reserved utilization (= latency bound)
- Need to be able to reorder requests for performance
  - All requests that can be handled without jeopardizing deadlines are put into a reordering set
- Cannot ignore "context switches" (seeks)

Fahrrad: RAD-based I/O scheduling

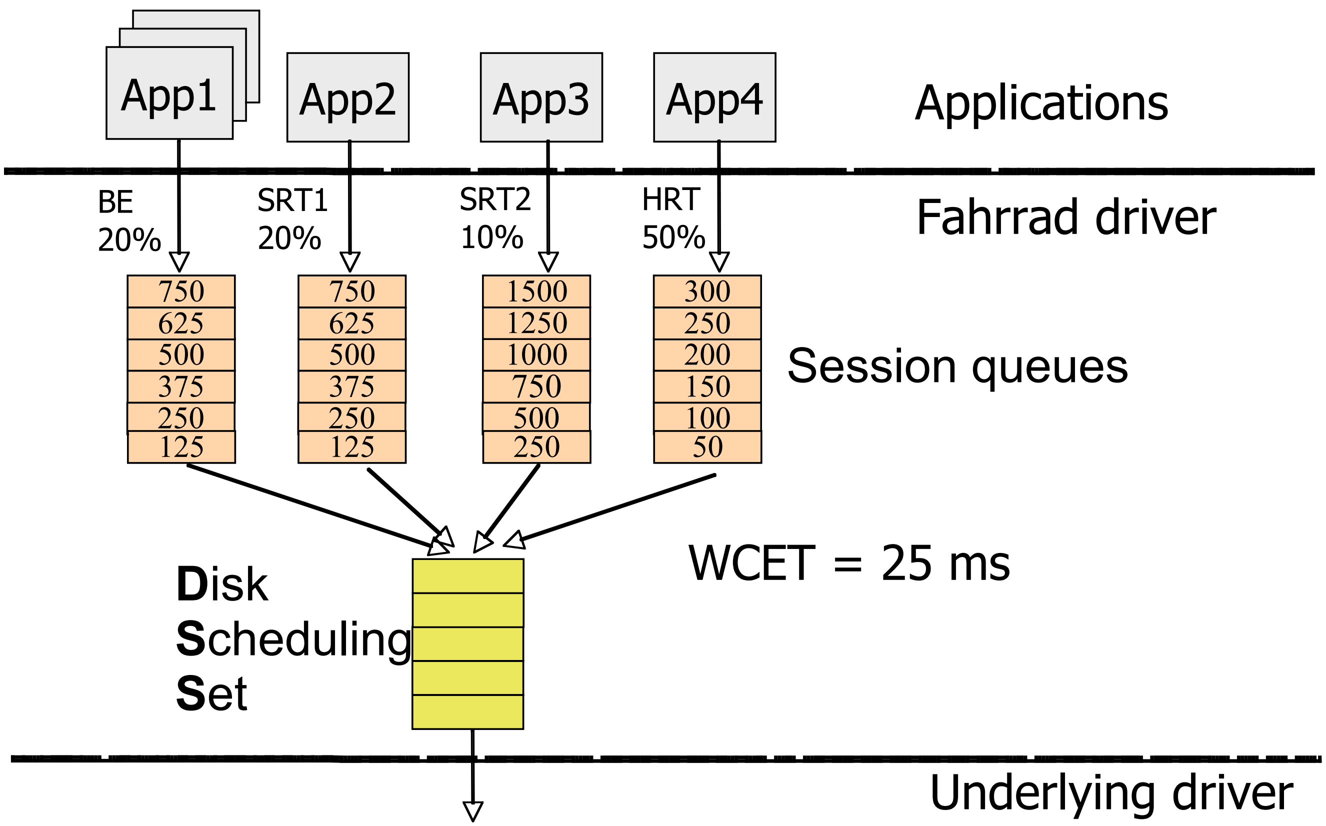
1. Utilization-based reservation, with deadlines
   - e.g., 50% of the disk every second, 10% every hour, etc.
2. Requests put into queues
   - Each queue has a rate and deadlines
3. Micro-deadlines assigned to requests based on target rate and worst-case assumptions
4. Requests released to Disk Scheduling Set (DSS) based on micro-deadline
5. Requests scheduled for service from DSS
6. Micro-deadlines updated based on actual service times

Fahrrad architecture



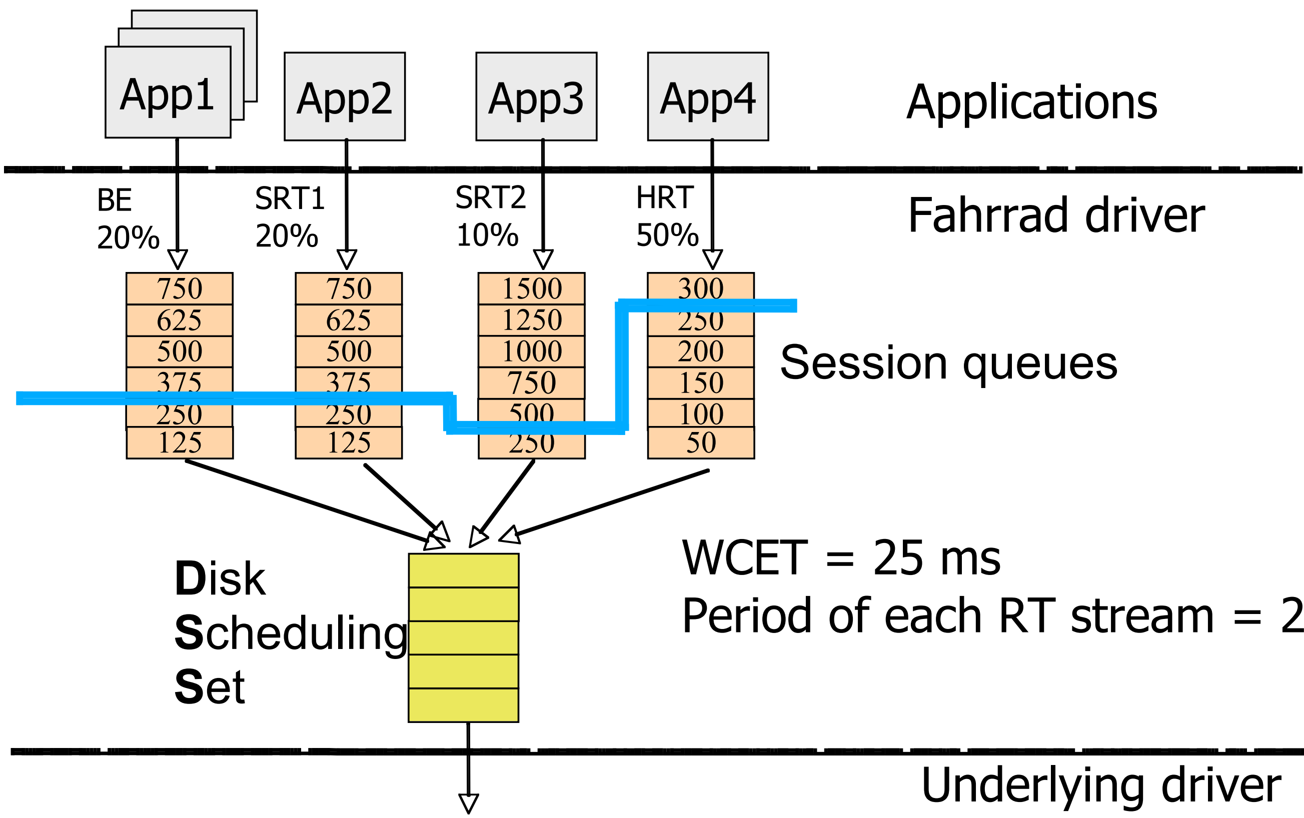
Guaranteeing deadlines

µdeadlines assigned to each request: $d_i = d_{i-1} + WCET / U$



Release to DSS

Requests with μ-deadline up to horizon (earliest deadline) move to DSS

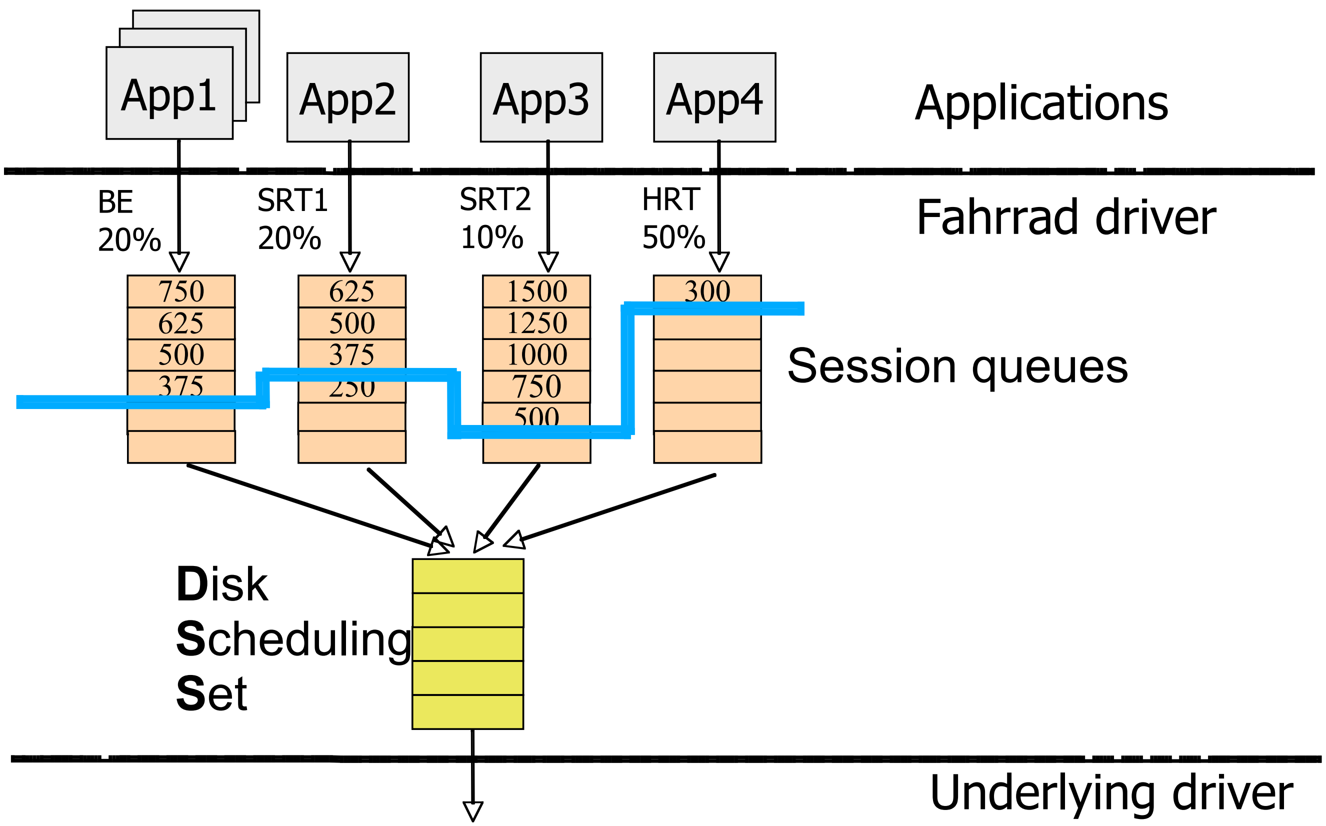


WCET = 25 ms

Period of each RT stream = 250 ms

Guaranteeing utilization

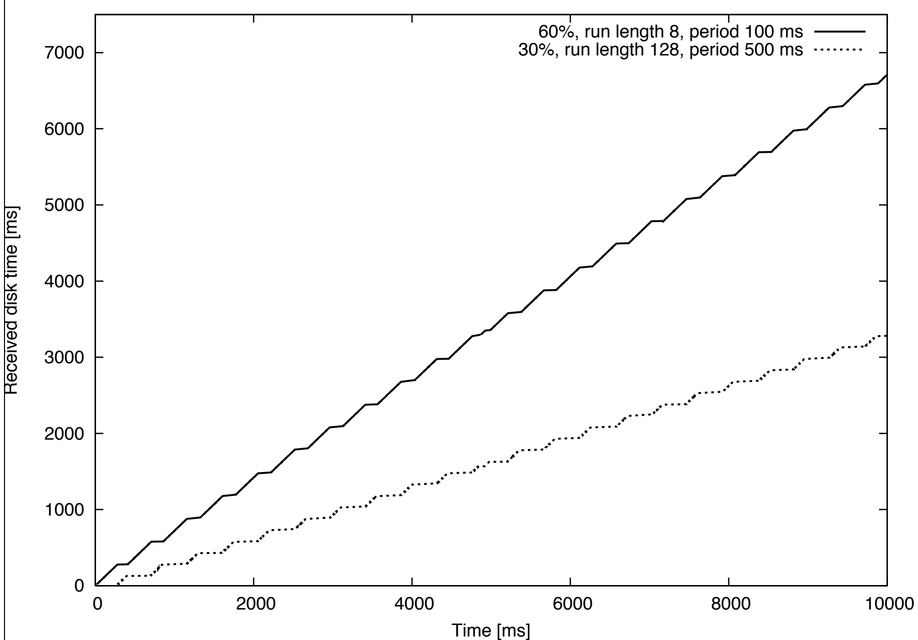
Guarantee reserved utilization by shifting μ-deadlines



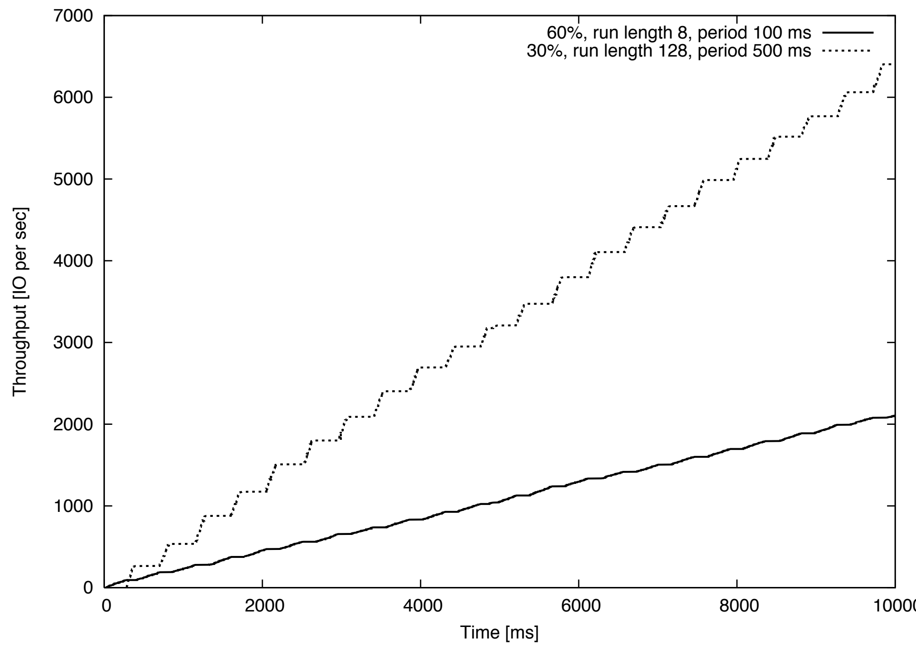
A few details

- DSS scheduling
  - C-SCAN, SPTF, **EDF**
- Managing burstiness
  - *Slots*—reserve utilization until request arrives
    - Unused slots are allocated to other streams
  - *Slot swapping*—aggregate requests in DSS by swapping slots
    - Increases sequentiality of DSS
    - Increases isolation and performance
- Isolation—accounting for overheads
  - Each stream charged for its seeks
  - Each streams charged 2 seeks per deadline

Fahrrad works



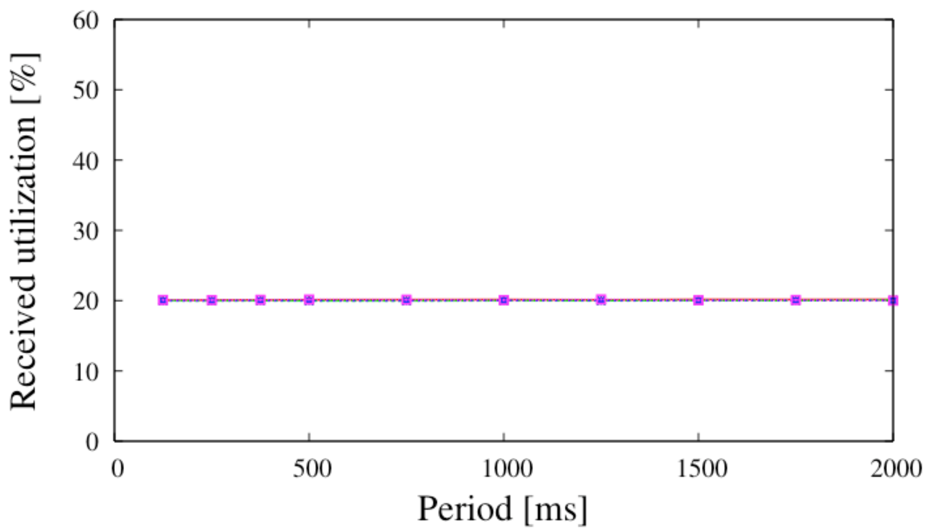
Utilization



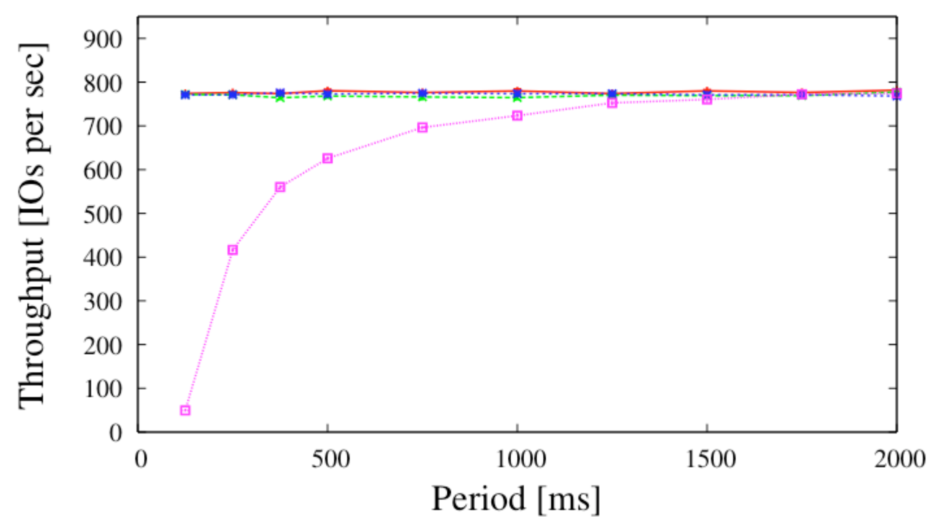
Throughput

Isolation between request streams

Utilization



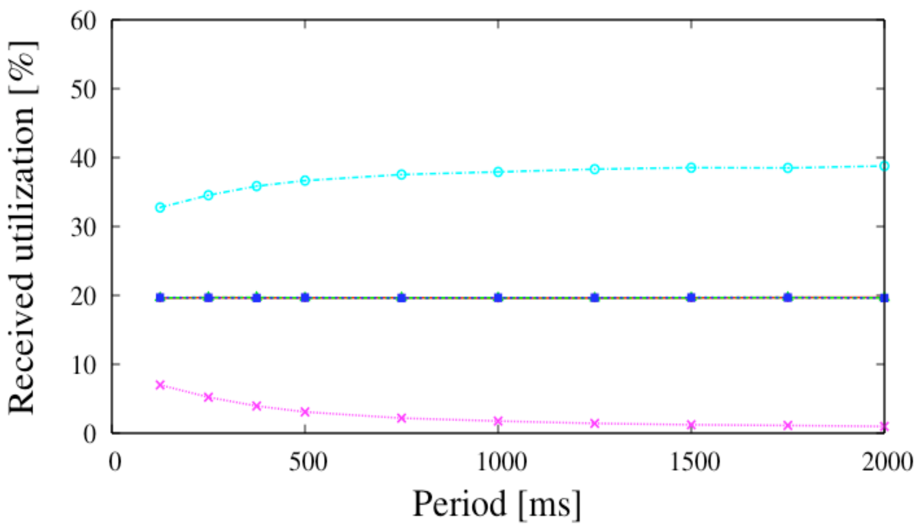
Throughput



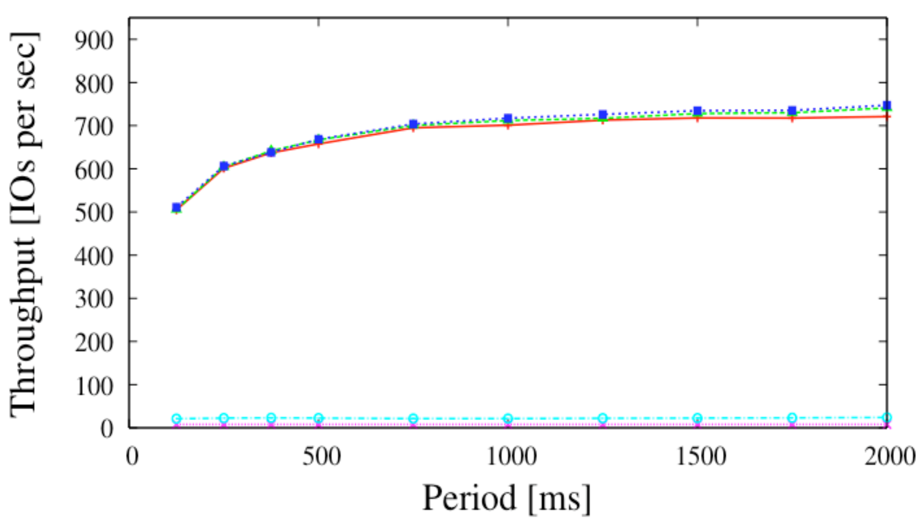
- Utilization and throughput of 4 I/O streams as period of stream 4 changes (sequential streams w/long queues)
- Rate: 20%
- Deadlines
  - Streams 1-3: 2s
  - Stream 4: varies from 125 ms to 2 s

HRT and BE (slack goes to BE)

Utilization

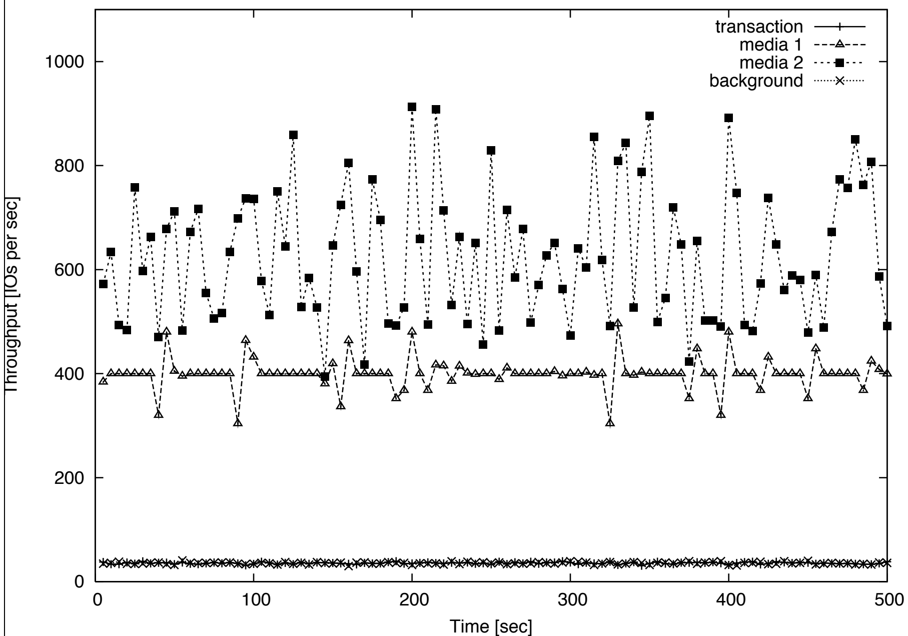


Throughput

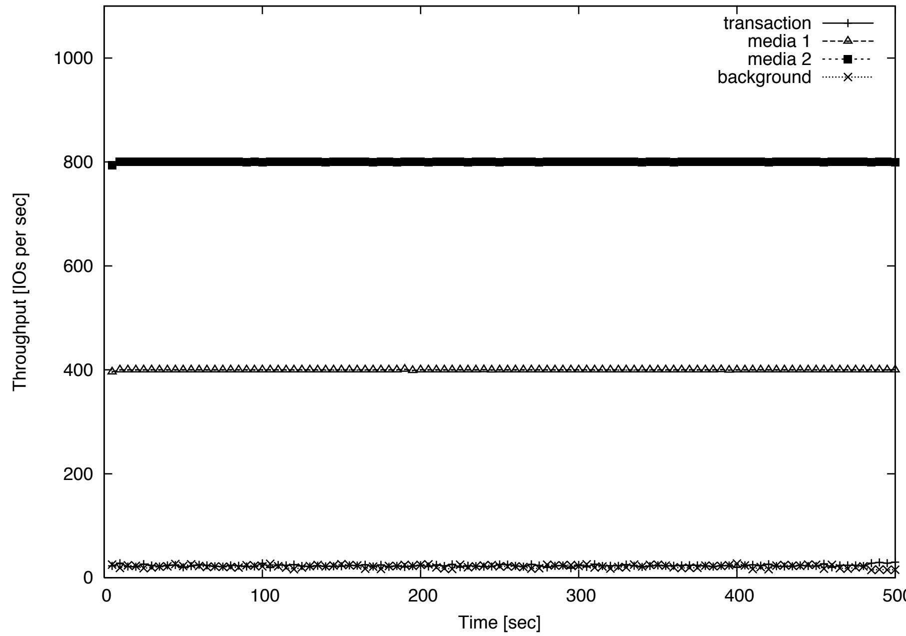


- Utilization and throughput of I/O streams as period of stream 4 changes
- Rate: 20%
- Deadlines
  - Sequential SRT streams & random BE stream: 2s
  - HRT: varies from 125 ms to 2 s

Performance vs. Linux



Linux



Fahrrad

Disk scheduling conclusions

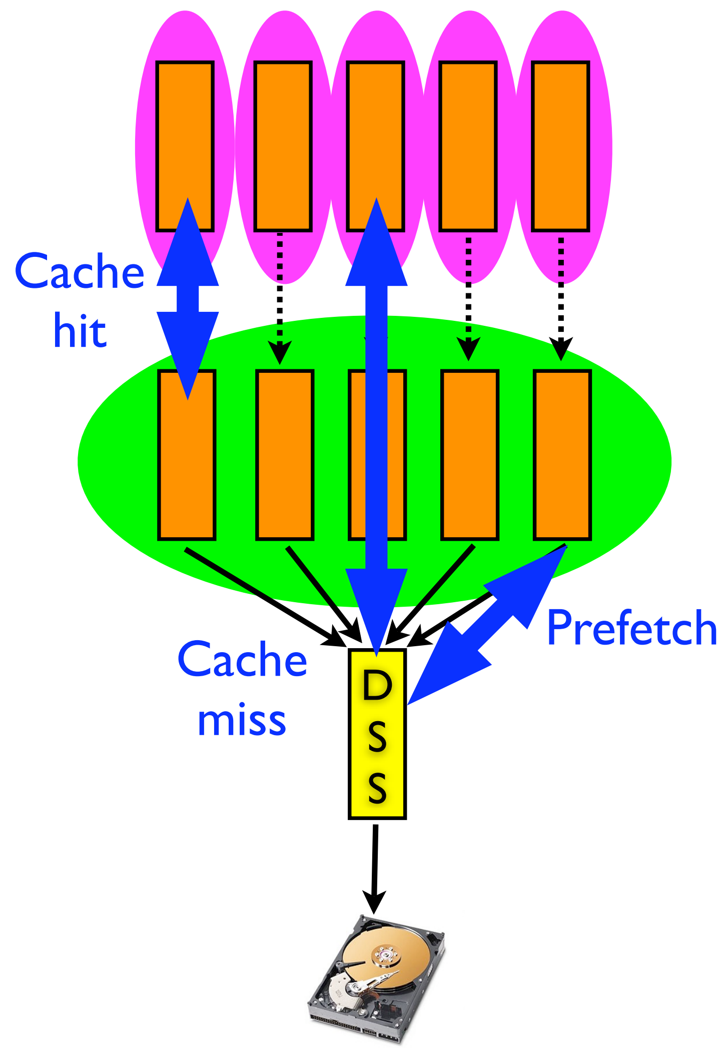
- Fahrrad provides
  - Integrated hard real-time, soft real-time, and best-effort service
  - Arbitrary (nearly) reservation granularity
  - Excellent isolation between processes
  - Excellent performance

Server cache management

- Server caching isolates disk from application behavior
  - Buffering smooths workload
  - Isolates disk from application period
    - Disk deadlines are buffer full times
  - Translates between time to space (and back)
  - Aside: best-case for disk = worst-cast for cache

Server cache management

- Reads and writes are handled differently
- Read cases
  1. *Cache hit*: creates slack
  2. *Cache miss*: sent to disk
  3. *Prefetch*: uses slack to increase efficiency
- NV cache⇒writes can be delayed indefinitely
- In general: need at least 3 periods of server cache



Network management

- Moving data from client cache to server cache
- Network QoS is well-explored
  - Currently examining existing solutions
- Cases
  1. One client/server route: O(1)
  2. One client/server route with arbitrary application placement: O(n)
  3. Many client/server routes
     - w/trunking: polynomial with linear programming: O(n)?
     - w/out trunking: NP-complete?

Client cache management

- Holds application data for transfer to server
- Further isolates application from disk
  - Further reduces burstiness
  - Further addresses independence of periods
- Coordinates with network and server cache

Spinoff: virtual disks

- Virtual disks—complete isolation of disk functionality
  - Capacity isolation
  - Temporal isolation
  - Performance isolation
- LUNs provide capacity isolation
- Fahrrad provides temporal and performance isolation

Conclusions

- Excellent progress (< 1 year along)
- Disk scheduling: Fahrrad
- Server cache: In progress
- Networking: Preliminary investigation
- Client cache: TBD
- Lots of industry interest: IBM, NetApp, VMware, SAP, NICTA/OK Labs, ...
- Pursuing DARPA follow-on building on end-to-end QoS